Towards a Middleware for Generalised Context Management

Richard Glassey, Graeme Stevenson, Matthew Richmond, Paddy Nixon, Sotirios Terzis, Feng Wang, Ian Ferguson

Global and Pervasive Computing Group
Department of Computer and Information Sciences
The University of Strathclyde
Glasgow, Scotland.

{firstname.lastname}@cis.strath.ac.uk

Abstract

It is widely accepted in the Pervasive Computing community that contextual interactions are the key to the delivery of truly calm technology. However, there is currently no easy way to incorporate contextual data into an application. If contextual data is used, it is generally in an ad hoc manner, which means that developers have to spend time on low-level details. There have been many projects investigating this area, however as yet none of them provide support for all of the key issues of dynamic composition and flexible representation of contextual information as well as the problems of scalability and adaptability to environmental changes. In this paper we present the Strathclyde Context Infrastructure (SCI), a middleware infrastructure for discovery, aggregation, and delivery of context information.

1. Introduction

Pervasive computing can broadly be defined as *calm technology* that delivers the correct service to the correct user, at the correct place and time, and in the correct format for the environment with minimal user distraction. It is widely assumed that the introduction of technology can always bring benefits to the end user. However, the user's interactions with technology are usually context-free (for example, the location, history, network behaviour and loading, and user preferences are not used to tailor the interaction). This results in largely homogenous presentation of service and interaction to the user, that in turn results in often cumbersome, and sometimes unusable, services. It is widely accepted in the Pervasive Computing community that **contextual** interactions are the key to the delivery of truly *calm technology* [1].

One demanding challenge for pervasive computing is how to collect and process context data from sensors and other sources. Most early researchers built their solution in an ad hoc way to investigate the problem space [2,3]. They had to consider everything, including the details of reading sensor data, distributing sensor data, and transforming sensor data into high-level data as well as application adaptation behavior. From a software engineering perspective, this makes developing applications for pervasive computing very cumbersome. Instead of exploring the potential of pervasive computing, designers have to spend time on low-level technical details. Although recent projects have shown progress in this area (see Section 3), no single proposed solution covers all aspects of the problem.

In this paper we present an infrastructure for contextual services. The focus is on the extraction, placement, and management of context in the face of mobility. We premise our work on an overlay network view that more accurately mirrors the way pervasive services will be deployed and used. We identify core concepts, such as our notion of range, which abstracts over the varying network and sensing technologies, while still providing appropriate programming models and abstractions.

The paper is structured as follows: Section 2 provides a summary of related work focused on systems issues in context management; Section 3 describes the principles underlying our architecture, introducing the core concept of **range** and our **models of location, mobility and composition**; Section 4 gives a detailed view of aspects of the implementation; Section 5 provides a sample application of the system; and Section 6 provides a view of the potentials and future directions of the infrastructure.

2. Related Work

Realizing the limitation of an ad hoc solution, Dey et al developed the Context Toolkit [4] to simplify the collection and processing context data. In the framework, there are three kinds of components: widgets, aggregators, and interpreters. The Context Toolkit provides common functionality such as communication between context components and encoding of context data. Consequently application developers only need to add sensor-specific code for each sensor and tailor context-dependent processing code.

At design time, the developer has many choices when he decomposes the task of gathering and processing context information into a set of widgets, aggregators and interpreters. But after the decision has been made and these context components are built, they become fixed. This means that the developer has to foresee all the requirements of applications at design time, which is unrealistic in pervasive computing environments where changes are frequent.

Chen and Kotz have proposed Solar [5], an infrastructure for collecting and aggregating data in ubiquitous computing environment. In their infrastructure, all the communication between context components is through events. Solar supports dynamic composition of context components; where dynamic composition results from changes in the environment. Besides the system-provided context components, it also supports using user-provided context components. It requires the application developer to explicitly specify the composition graph of context components. The infrastructure will try to find the common parts of context processing graphs of different applications and will reuse them, thus improving scalability.

Since the foci in Solar are scalability and flexibility, they have not addressed the issue of robustness. In pervasive computing, the same context may come from several sources and the data sources may become available or unavailable due to user movement or component failure. The requirement that the application developer has to explicitly choose data source, context operators and specify the context-processing graph will affect the robustness of the context system.

iQueue from Cohen et al is a pervasive data composition framework. The iQueue framework "enables applications to focus on the semantics of composition by facilitating the mechanics of composition" [6]. iQueue aims to help application designers by handling the heterogeneity and diverse distribution of data sources that are available for processing. An iQueue application obtains its data from composers. A composer combines data sources to produce a particular result. Data sources are described by data specifications, which are descriptions of data type required by the composer, rather than explicitly where to find the data. A composer both produces values that it computes and accepts subscriptions for notification of new values. iQueue supports the continual rebinding of data specifications to the most appropriate data sources in order to make the best use of data available to the application at any given time.

It is widely accepted that relevant contextual information may take numerous forms but ultimately tell us the same thing. iQueue faces this issue when presented with data sources that have widely different syntactic descriptions but are semantically similar. For example an iQueue application that has been developed to request location data from a network of door sensors cannot take advantage of an environment that provides location information using a wireless detection scheme.

Garlan et al have taken a novel approach to managing pervasive computing environments with Project Aura [7, 8]. Aura aims to "minimize distractions on a user's attention, creating an environment that adapts to the user's context and needs" [7]. The two broad concepts that Aura wishes to achieve are *pro activity* and *self-tuning*. Firstly Aura should pick the most suitable mode of interaction for the current task whilst changes in modes of interaction should appear seamless. Secondly, Aura is context aware, i.e. it should know about its environment and take appropriate actions e.g. hide sensitive information. Thirdly Aura has a notion of predicting what task the user is trying to achieve. This is likely to be the most difficult task within the project and will require more than a technological solution. Aura proposes an infrastructure that moulds itself to the user's task or needs with little need for user intervention. It appreciates that human attention is a limited resource especially in mobile situations. It however does not clearly address the problems of scalability that are inherently important in pervasive computing environments.

While investigating the above projects, we have observed the following open issues in the management of pervasive computing environments that we wish to address with our infrastructure:

- Dynamic composition of context entities;
- Control over the quality and structure of context composition;

- Adaptivity to environmental changes (e.g. component failure);
- Flexible and extensible representation and retrieval of contextual information; and
- Scalable infrastructure.

3. SCI Architecture

The **Strathclyde Context Infrastructure (SCI)** is organised into two distinct layers. The upper layer of the infrastructure is a network overlay of partially connected nodes and is referred to as the **SCINET** (see Figure 1). The lower layer of the infrastructure concerns the contents of each node, which consists of **entities** (People, Software, Places, Devices and Artifacts) responsible for producing, managing and using contextual information, and is referred to as a **range**.

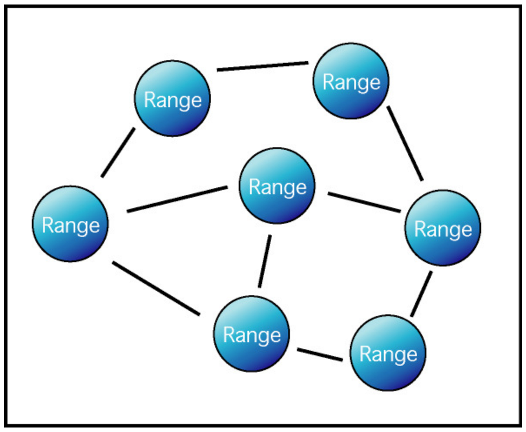

Figure 1: SCINET

The SCINET is concerned with managing interactions that take place between two or more ranges in order to provide appropriate contextual information. It is likely entities that exist in one range may be interested in consuming contextual information from entities in other ranges.

The network overlay approach provides the infrastructure with favourable scalability and robustness characteristics that would have not been possible with a hierarchical arrangement of nodes. Routing through an overlay network avoids any bottlenecks created when using hierarchical infrastructures whilst achieving comparable performance [9]. It also provides the necessary level of abstraction in order for entities to communicate across many heterogeneous network types using GUIDs rather than traditional addressing schemes. The SCINET can be created via Range discovery, requiring little initialisation. Alternatively it may be desirable to group relevant Ranges together, such as those operating within an individual building or across a larger area in order to control access and increase performance.

A Range is defined as an area that can be described in logical and/or physical terms. A Range can be can be bounded by a physical area (a collection of adjacent rooms, an entire floor of a building) or by the effective operating range of a particular network type (e.g. a wireless network). By this definition both physical entities, such as doors and rooms, network availability and software components can be represented under the same common model.

Each Range is governed by it's own individual **Context Server** (CS), the hub for the Range. A CS is considered to be a secure, always on central server for management of contextual information within a Range. We believe that the complexity and timely response required when providing contextual information justifies the use of a centralised service. This approach moves the responsibility regarding the capturing and processing of contextual information away from the entities that exist within the infrastructure.

3.1 Structure of a Range

This section demonstrates the mechanisms for managing and providing contextual information within a Range. Each Range contains a single Context Server, which manages three types of components; Context Entities, Context Utilities and Context Aware Applications (see Figure 2).

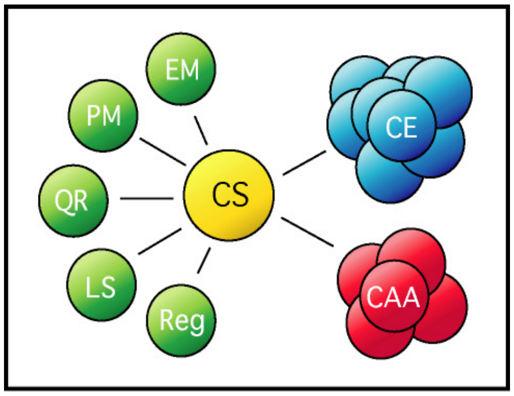

Figure 2: Range

The **Context Server** (CS) is the most important component of a Range. It manages the other components and provides the means of communicating with other Ranges in the SCINET. It maintains a central store of entity information as well as managing the context utilities operating within its range. The CS provides the access point for Context Aware Applications to interact with the infrastructure.

A **Context Entity** (CE) is a lightweight software component for representing an entity within the infrastructure. A CE allows its entity to communicate by means of producing and consuming typed events. This abstraction allows non-computational entities to be included within the scope of the infrastructure. A CE maintains a **Profile** for its entity that contains meta-data describing the entity. For entities that provide a service, the CE may also maintain an **Advertisement** describing the services that this entity can provide to other entities. All CE's are registered within a range when they arrive and deregistered upon departure. While active within a Range, the Range's Context Server manages both the CE's Profile and Advertisements.

A **Context Aware Application** (CAA) is an application that has the ability to pull or be pushed contextual information to or from the infrastructure. A CAA communicates with the CS by way of a **Query** as described later. Indeed the CAA may itself provide additional information regarding its current context. For example, a CAA can make use of a users Profile stored in their CE to determine previous behaviour or preferences in order to provide a more useful service.

The **Context Utilities** (CU) are set of specialist services that help the CS in the management of a Range. Whilst there is possibility for many types of CU, a core set exists in all Ranges.

- *Range Service*: Responsible for detecting the arrival into and departure of entities from a Range.
- *Query Resolver*: Provides the means to take a high level query and decompose it into a useful configuration of Context Entities.
- *Location Service*: Handles the resolution of location related tasks.
- *Profile Manager*: Provides access and update abilities to Context Entities Profiles.
- *Event Mediator*: Manages the establishment, maintenance and removal of event subscriptions between Context Entities and Context Aware Applications.
- *Registrar*: Maintains an accurate view of all entities within the current Range.

3.2 Model of Composition

In order to provide flexibility with regards to the provision of context, SCI has been designed to support data composition by means of forming **configurations**. A configuration is an event subscription graph between entities where the inputs to one CE are provided by the outputs of others. To achieve this, we use query data along with input and output information obtained from CE Profiles to perform type matching. When this process is complete, setting up subscriptions between CE's to their data sources creates the required graph. This feature of our infrastructure allows us to aggregate different types of contextual information in order to provide a more useful service to applications.

To illustrate a configuration, consider a CAA on a mobile device that displays a building floor map and can visually represent the path from one location to another. Now imagine the scenario where a user, Bob, wishes to display the path between himself and his colleague John. Normally, you would require a purpose built service to provide such information but our infrastructure facilitates the creation of a configuration from basic CEs to accomplish the same task. An outline of the steps involved is as follows:

- The query generated by the CAA (pathApp) requests the Path between Bob and John
- The query resolver used by the Context Server searches CE profiles for entities that provide path information as an output.
- A CE (pathCE) is found that meets this requirement but requires two locations as inputs.
- The query resolver then searches for CEs that are able to provide the locations of the required entities (John and Bob).
- An objLocationCE is found that takes an entity ID as an input and produces location information as an output. When this entity was added to the system it was set up to subscribe to all events emanating from door sensors (doorSensorCEs).
- The doorSensor CEs produce events indicating when an object (equipped with ID tag) passes through them

Once a complete configuration has been discovered (i.e. down to the sensor/data level) to fulfill a query's requirements, the Context Server sets up event subscriptions between the CEs involved (see Figure 3).

Because the required information is delivered to the CAA through a dynamic subscription graph, any changes in state (i.e. the generation of new events) will cause updated information to be delivered to the application. For example, if John walks through a door, the event generated by the doorSensorCE will cause the objLocationCE to dispatch a new event, which in turn will result in the pathCE dispatching a new event to the pathApp. This means that the pathApp will always have correct information regardless of environmental changes.

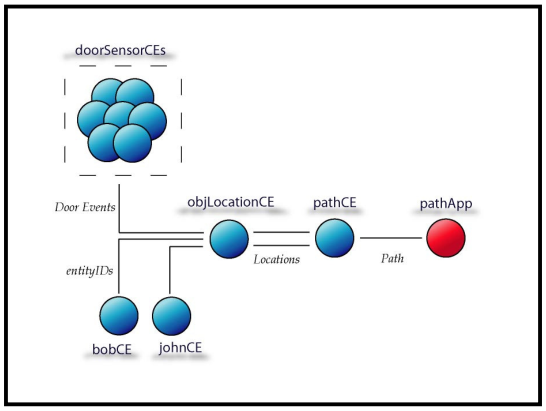

Figure 3: Configuration of CEs

3.3 Model of Location

As discussed earlier, a Range can be either described physically, logically or indeed both where necessary. This concept allows for a wider scope of contextual environments to be modelled instead of imposing a fixed model. Two contrasting examples of a Range are as follows. A collection of labs and offices forming a research area can be considered a Range. An open space within a campus, not restricted by walls or buildings is also a candidate for a Range.

It is inevitable that the choice of such an abstract means of representing the Range will have serious implications for the choice of an appropriate location model. With this in mind we propose that it is preferable to support many types of location model and interoperate between them if necessary. For example it may be necessary to convert geometric information to a hierarchical model or similarly convert network signal strength to a geometric position [7]. To facilitate this it will be necessary to develop an intermediate location language.

3.4 Model of Mobility

In a dynamic environment entities will move in and between Ranges throughout their lifecycle. To allow for this mobility each range monitors internal activity as well as activity at its boundaries in order to detect the arrival and departure of entities. For example, a user wearing an id tag arriving or leaving their range by walking through a door equipped with a sensor for detecting id tags would be discovered. Similarly a user with a W-LAN equipped device could be detected leaving the effective operating range of a wireless network.

4. Current Status

We are currently developing a prototype of the infrastructure using Java along with a hybrid communication model (a combination of distributed events and point to point communication) for entity interaction. CE Profiles consist of simple Metadata about entity inputs and outputs while Advertisements take the form of 'well known' interfaces in order that CAAs may transfer service specific data to CEs.

4.1 Component Design

Figure 4 shows the high level design of the CE's and CAAs. Both entities share the RegisterInterface in order to facilitate communication with a **Range Service** (see below) while CAA's include the ConsumeInterface for dealing with events (in response to a query). The ServiceInterface, implemented by the CE represents the 'well known' Advertisement interface discussed above. At the Concrete level, CE or CAA developers need only to deal with the service they provide or the events they receive. The work of integrating components into the system, query submission and event distribution is all handled internally by the infrastructure.

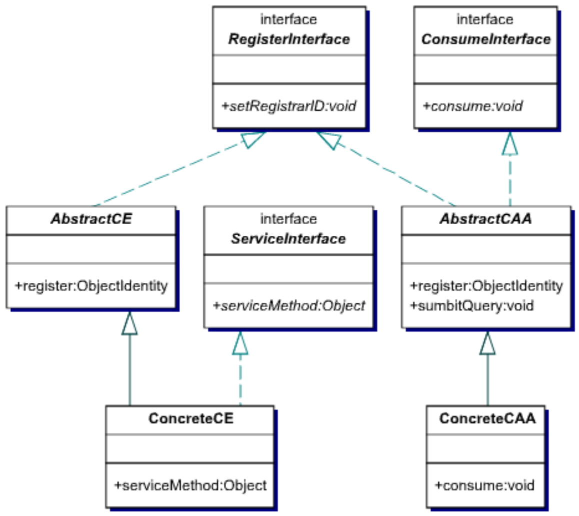

Figure 4: Architectural Design

4.2 Entity Discovery

An important aspect of a dynamic environment is the way in which components are detected and integrated. Figure 5 shows how this is done within the SCI infrastructure. When a Context Server starts up, it deploys a **Range Service (RS)** to all the machines within its jurisdiction. The RS performs the task of listening for CAAs or CEs starting up in order to inform them about the Range's **Registrar**. The CAA/CE can then contact the Registrar in order to gain access to the infrastructure. Upon completion of the registration process, the Registrar will return the Context Server details to a CAA (in order to submit queries) or the Event Mediator details to a CE (in order to publish events). As mentioned above, this interaction takes place at the abstract class level in order that the application designer need only deal with component functionality.

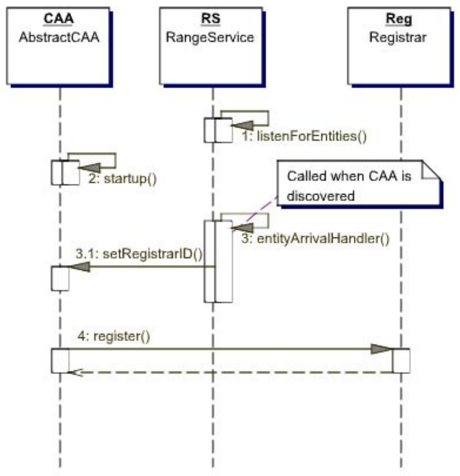

Figure 5: Discovery Sequence

4.3 Query Composition

Currently we use a simple query model to support requests for information from CAAs. A high level overview of this model can be seen in figure 6. There are five sections central to the formation of a query. The first four are as follows:

- *What*: Describes what this query is looking for, be it an entity type (e.g. a printer), a named entity (identified by a GUID) or information fitting a pattern (e.g temperature in degrees Celsius).
- *Where*: The location (if applicable) of the information required be it explicit (e.g. Room 10.01) or implicit (e.g. closest to me).
- *When*: The temporal aspect of the query, the conditions under which the configuration should be executed.
- *Which*: The desired qualitative aspects governing selection from multiple entities (e.g. shortest time to service completion) .

The final section, the mode, indicates the intent of the query. The infrastructure supports four types of query in order to meet possible user needs. These are:

- *Profile request*: In order to obtain information about CEs.
- *Event subscription*: To subscribe to a piece of information and be updated with any changes.
- *One-time subscription*: As above, but the subscription is cancelled after the CAA receives an event.
- *Advertisement request*: The interface to communicate with a service

```
<query>
        <query_id> </query_id>
        <owner_id> </owner_id>
        <what> </what>
        <where> </where>
        <when> </when>
        <which> </which>
        <mode> </mode>
</query>
```

Figure 6: Query Model

5. CAPA: An Example Implementation

To illustrate a use of our infrastructure, we present CAPA: a Context Aware Printing Application that, as the name suggests, uses the SCI infrastructure to aid users in printing documents.

Consider the scenario where Bob is traveling to work by train. Bob loads CAPA on his PDA, which informs him that he is currently not in a range. Bob uses the application to queue several print jobs and tells the application that he wishes the documents printed to the closest printer when he reaches Room L10.01 (his office). As he is not currently within a range, the application stores the query for future use.

Bob finally reaches the university and enters the Livingstone Tower. The network base station in the lift lobby detects Bob's PDA which is then registered with the infrastructure. Once CAPA is informed that it has a connection to SCI, the previously stored query is submitted to the lobby range's Context Server. The Context Server looks at the query and identifies that the query should be forwarded to the Context Server for Level Ten. Once this happens, the Level 10 Context Server analyses the query, builds a configuration (X) to provide an answer and stores it until its temporal constraints are satisfied. The context server then 'listens' for Bob entering L10.01.

When Bob reaches his office, the door sensor generates an event indicating that Bob has entered (Bob is wearing an electronic ID badge). The Context Server receives this event and proceeds to execute configuration X. Once complete, P1 is identified as the closest printer to Bob (see Figure 7) and CAPA is informed. CAPA then contacts the Context Entity for printer P1 and sends it the documents to print.

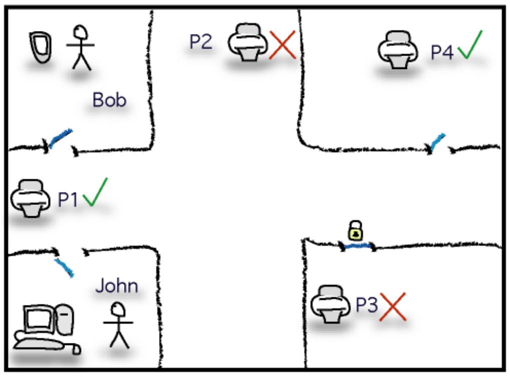

Figure 7: Printer Selection

Meanwhile, John, whose office is next to Bob's, wants to quickly print a document before he leaves to give a lecture. He loads the CAPA software on his computer and requests his document be printed to the closest printer with no queue. As before, the Context Server builds a configuration (Y) to answer the query. During execution, the following things are found:

- Printer P1 is currently being used by Bob
- Printer P2 is unavailable due to being out of paper
- Printer P3 is behind a locked door to which John has no access

Printer P4 is identified as being the closest free printer with no queue and CAPA is informed and proceeds as before. John can now pick up his printout and get to his lecture on time.

6. Conclusions and Future Work

We propose a comprehensive infrastructure to simplify the development of context-aware applications. Applications can use the provided context query language to express their context requirements. The infrastructure will compose the context processing components and data sources automatically in order to deliver the required context. It will also adjust the composition of these components dynamically in the case of environment changes, thus improving service and fault tolerance while minimising user intervention. Although only in the early stages of development, many issues have arisen that will require further study:

1. The topology, and management, of the overlay network and the consequent placement of computation and data for timely response to events [9];
2. The structure and form of a context query language which encapsulates composition operators, notions of semantic equivalence, partial equivalence, relevance and contracts on quality of the context information;
3. Autonomic behaviour has to be supported. This requires the use of appropriate architectural description mechanisms [10] and adaptation approaches. We consider the implications of providing bounds on acceptable adaptation, the scalability of adaptation in the face of large number of adapting entities and the overall stability of the system as critical;
4. Appropriate location models that capture the geometric, topological, and logical spatial relations have to be developed to allow fine grained control over the interaction of entities with the real world and the user;
5. Programming primitives have to be developed that allow the description of behaviours appropriate for pervasive systems. Such primitives will have to capture notions of partial failure, quality of information, availability of service, capabilities, user preference, domain of interaction, and mobility.

In conclusion, to enable the exploration and development of pervasive computing systems we have presented here the development of a general context gathering and management infrastructure. We describe an open source infrastructure that supports context gathering and storage.

Acknowledgements: This work is supported by the EU FP5 GLOSS Project (IST-2000-26070), in collaboration with Trinity College Dublin, Université Joseph Fourier, and The University of St Andrews, and through research student scholarships from the University of Strathclyde.

References

[1] Mark Weiser, *Some Computer Science Issues in Ubiquitous Computing*. Communications of the ACM 36(7): 74-84 (1993).

[2] Long, Sue, Rob Kooper, Gregory D. Abowd and Christopher G. Atkeson, *Rapid prototyping of mobile context-aware applications: The Cyberguide case study*. In the Proceedings of the 2nd ACM International Conference on Mobile Computing and Networking (MobiCom '96), pp. 97-107.

[3] Abowd, Gregory D., Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper and Mike Pinkerton. Cyberguide, *A mobile context-aware tour guide.* ACM Wireless Networks 3(5): pp. 421-433. October 1997.

[4] Anind K. Dey, Daniel Salber and Gregory D. Abdowd, *An Architecture to Support Context-Aware Applications.* Submitted for review to UIST' 1999.

[5] Guanling Chen and David Kotz, *Supporting Adaptive Ubiquitous Applications with the Solar System.* Technical Report TR2001-397, May, 2001.

[6] Norman H. Cohen, Apratim Purakayastha, Luke Wong, and Danny L. Yeh, *iQueue: a pervasive data-composition framework.* The 3rd International Conference on Data Management, Singapore, Janauary 8-11, 2002.

[7] David Garlan, Daniel P. Siewiorek, Asim Smailagic and Peter Steenkistie, *Project Aura: Toward Distraction-Free Pervasive Computing.* IEEE Pervasive Computing Magazine, April-June 2002.

[8] Joao Pedro Sousa and David Garlan, *Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments.* 3rd Working IEEE/IFIP Conference on Software Architecture, Montreal, 2002.

[9] A Dearle, G.N.C Kirby, R Morrison, A McCarthy, K Mullen, Y. Yang, R.C.H Connor, P. Welen, and A. Wilson, *Architectural Support for Global Smart Spaces.* LNCS 2574, Proceedings of the 4th International Conference on Mobile Data Management, Springer Verlag, pp 153-164, 2003.

[10] P A Nixon, S Terzis, F Wang, S A Dobson, *Architectural Implications for Context Adaptive Smart Spaces*. Proceedings of IEEE International workshop on Networked Appliances, pp 156-161. 2002. IEEE Press.